University of Bordeaux
Year 2015-2016

Master ESSV
MtoCPL

TD n°6 - Existential woes

▷ Java and C# propose two different ways to define constraints on the variables :

- Java/ Use-site variance : the variance indication on the type variable appears in the method using the variable

```
class Collections<T> { .. };
public void copy(List <? super T> dst, List <? extends T> src) {
.. };
```

- C#/ Definition-site variance : the variance indication on the type variable appears at the definition of the class

```
interface Collections<in T> {
  T getElem()
.. };
```

In both cases, the code must satisfy the *get-put* principle for a generic class `C<T>` :

- if the type variable `T` is in a *covariant* position, one can call every method in `C<T>` except the generic methods having an *argument* of type `T`.
- if the type variable `T` is in a *contravariant* position, one can call every method in `C<T>` except the generic methods having a *result* of type `T`.

Exercice 1: Type variance

In languages with generics, the `Collections` and their containers provide nice examples of parameterized types.

1. What is in general (that is to say without taking a specific language into account) the rule of variance for a parameterized type such as `Container<Parameter>` ?
2. Verify your proposition by writing examples in Java and C#. What is the error message, if any, and when does it occur ?

According to the previous questions, if ever an `Orange` inherits from a class `Fruit`, then it implies that there is no relation whatsoever between `List<Orange>` and `List<Fruit>`, which is very restrictive.

3. Give an example that shows how restrictive this choice is.

Now there is a possibility to alleviate this behavior with variance indications. Type compatibility is explicitly stated, either at the definition of the variables (*definition-site variance*, for example in C# and Scala) or at the place they are used, usually the arguments of functions (*use-site variance*, for example in Java via the use of *wildcards*).

4. Consult the Java 1.7 documentation page for the `Collections` class (http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html), and discuss the types of the functions `fill`, `copy` and `max`.

5. Consult the C# documentation for the `IEnumerable` interface http://msdn.microsoft.com/en-us/library/vstudio/9eekhta0%28v=vs.100%29.aspx and discuss the differences with the Java case, in particular for the `max` functions.

For these specific types, there is a general principle explaining what kind of methods can or cannot be applied to a wildcard type. For each of the following citations[1], construct a Java example that does not type-check, and explain why it doesn't.

6. About covariance :

> ▷ For example the type `List<? extends Number>` is often used to indicate read-only lists of `Number`s. This is because one can get elements of the list and statically know they are `Number`s, but one cannot add `Number`s to the list since the list may actually represent a `List<Integer>` which does not accept arbitrary `Number`s.

7. About contravariance :

> ▷ Similarly, `List<? super Number>` is often used to indicate write-only lists. This time, one cannot get `Number`s from the list since it may actually be a `List<Object>`, but one can add `Number`s to the list.

Let us write a generic function in to sort elements of a list.

8. Write the prototype of a generic sorting function in C# that takes a list and a comparison function, that is covariant in the list parameter and contravariant in the other.

[1] Taken from http://www.cs.cornell.edu/~ross/publications/tamewild

9. Write the prototype of a sorting algorithm in Java that is equivalent to the previous one. The functional comparison can be replaced by the `Comparator` interface.

   What is the difference with the C# solution ?

10. Suppose that we only require that the type `T` implements the `Comparable` interface. What is the most generic prototype you can write ?